Problem 1

Calculate the Lagrangian $\mathcal{L}(x, \alpha)$.

$$\mathcal{L}(x, \alpha) = -(x_1 + x_2) + \alpha_1(x_1^2 + x_2^2 - 1)$$

Obtain the Lagrangian dual function $g(\alpha)$.

$$\begin{aligned} \frac{\nabla_{\mathcal{L}(x,\alpha)}}{\partial x_1} &= 0 \Leftrightarrow \\ 2\alpha_1 x_1 - 1 &= 0 \Leftrightarrow \\ x_1 &= \frac{1}{2\alpha_1} \end{aligned}$$

$$\begin{aligned} \frac{\nabla_{\mathcal{L}(x,\alpha)}}{\partial x_2} &= 0 \Leftrightarrow \\ 2\alpha_1 x_2 - 1 &= 0 \Leftrightarrow \\ x_2 &= \frac{1}{2\alpha_1} \end{aligned}$$

Solve the dual problem (Plug x^* in $g(\alpha)$).

$$\begin{aligned} \frac{\nabla_{g(\alpha)}}{\partial \alpha} &= 0 \Leftrightarrow \\ \frac{1}{\alpha_1^2} - \frac{1}{2\alpha_1^2} - 1 &= 0 \Leftrightarrow \\ \alpha_1^2 &= \frac{1}{2} && \text{(by constraint } \alpha_i \geq 0) \\ \alpha_1 &= \frac{1}{\sqrt{2}} \end{aligned}$$

Problem 2

- Similarities
  - Both algorithms try to solve the problem of binary classification by finding a decision boundary $w^\intercal x + b = 0$ that separates all datapoints x_i with label 1 from all datapoints x_j with label -1
- Differences
  - SVM has a closed form solution
  - SVM gives an unique solution (constrained optimization) by choosing a decision boundary s.t. it has a maximum margin to its nearest datapoints
  - Perceptron must be solved iteratively
  - Perceptron may have infinitely many correct solutions (if available) (unconstrained optimization)

Problem 3

By the formulation of the SVM problem we have

$$\begin{aligned} \text{minimize} \quad & f_0(w,b) = \frac{1}{2} w^\intercal w \\ \text{subject to} \quad & f_i(w,b) = y_i(w^\intercal x_i + b) - 1 \geq 0, \quad \text{for i = 1, ...,N} \end{aligned} \tag{1}$$

Clearly we can rewrite (1) to

$$f_i(w,b) = -y_i(w^\intercal x_i + b) + 1 \leq 0, \quad \text{for i = 1, ...,N}$$

By Slater's constraint qualification we have that the duality gap of the SVM problem is zero if $f_0(x)$, $f_1(x)$, ... $f_N(x)$ are convex and the constraints $f_1(x)$, ... $f_N(x)$ are affine. Clearly both assumptions are met, because $f_0(x)$ is simply the L_2-norm, which is convex and the constraints are linear functions in w shifted by an offset b, which makes them affine. Thus the duality gap is zero.

Problem 4

a). Let $\mathbf{X} \in \mathbb{R}^{n \times d}$, where each row is a datapoint $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{Y} \in \mathbb{R}^{n \times n}$, where each column is the label y_i of the i-th datapoint replicated n times and $\boldsymbol{\alpha} \in \mathbb{R}^n$ with α_i at position i.
By the Hadamard product we have

$$\mathbf{Q} = -XX^\intercal \odot (Y \odot Y^\intercal) \tag{1}$$
$$g(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\intercal \mathbf{Q} \boldsymbol{\alpha} \tag{2}$$

b). We can reformulate (2) to $Q = -(X \odot Y)^\intercal (X \odot Y)$ and define $A = (X \odot Y)$. By construction we have $A^\intercal A$ is positive semi-definite, i.e. $\forall z : z^\intercal A^\intercal A z \geq 0$ because $z^\intercal A^\intercal A z = (Az)^\intercal A z \geq 0$ (i.e. L_2 norm is non-negative). Thus $Q = -A^\intercal A$ is negative semi-definite.
c). From negative semi-definiteness of Q and the Hessian of $\alpha^\intercal Q \alpha$ is negative, it follows that $\alpha^\intercal Q \alpha$ is a concave function. Since in the dual formulation we maximize $g(\alpha)$, $\frac{\nabla_{g(\alpha)}}{\partial \alpha} = 0$ is a sufficient condition to get the global maximum α^*.

07_homework_svm-2

December 10, 2017

1 Programming assignment 7: SVM

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        from sklearn.datasets import make_blobs

        from cvxopt import matrix, solvers
```

1.1 Your task

In this sheet we will implement a simple binary SVM classifier.

We will use `CVXOPT` http://cvxopt.org/ - a Python library for convex optimization. If you use `Anaconda`, you can install it using

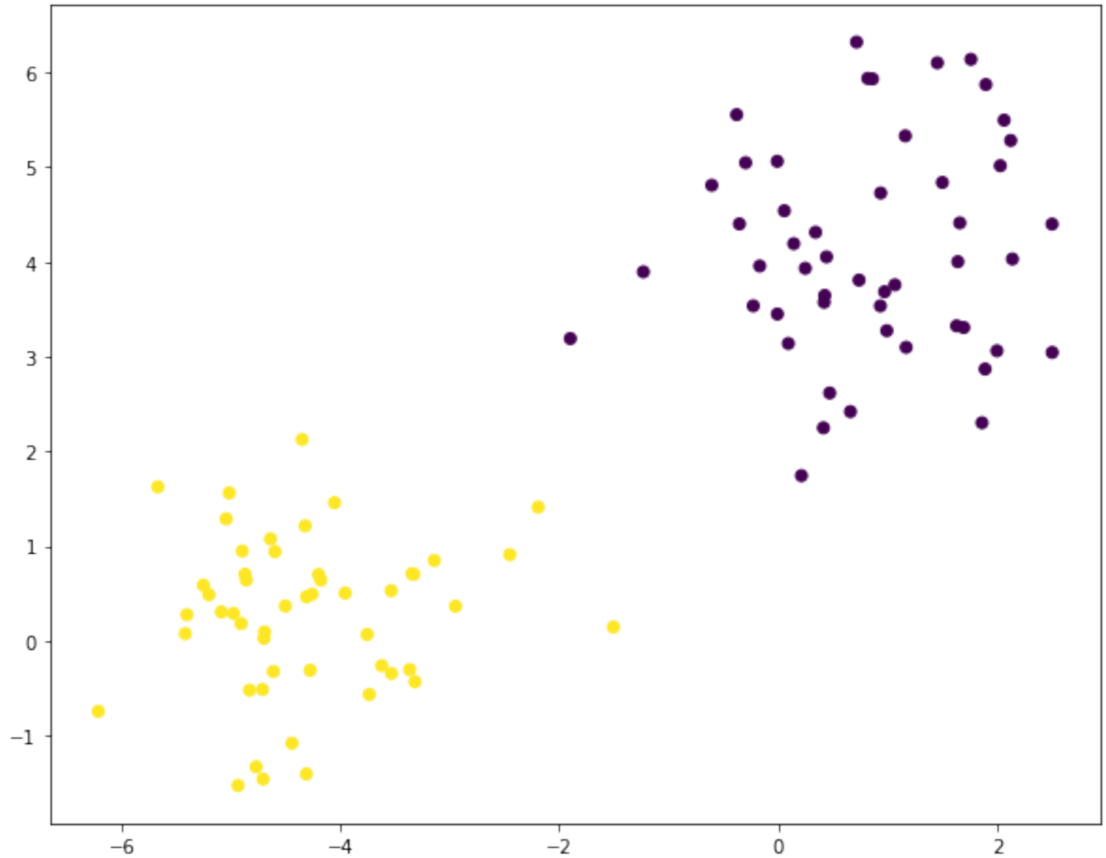
```
conda install cvxopt
```

As usual, your task is to fill out the missing code, run the notebook, convert it to PDF and attach it you your HW solution.

1.2 Generate and visualize the data

```
In [2]: N = 100  # number of samples
        D = 2  # number of dimensions
        C = 2  # number of classes
        seed = 3  # for reproducible experiments

        X, y = make_blobs(n_samples=N, n_features=D, centers=2, random_state=seed)
        y[y == 0] = -1  # it is more convenient to have {-1, 1} as class labels (instead of {0
        y = y.astype(np.float)
        plt.figure(figsize=[10, 8])
        plt.scatter(X[:, 0], X[:, 1], c=y)
        plt.show()
```



1.3 Task 1: Solving the SVM dual problem

Remember, that the SVM dual problem can be formulated as a Quadratic programming (QP) problem. We will solve it using a QP solver from the `CVXOPT` library.

The general form of a QP is

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x}$$

$$\text{subject to} \quad \mathbf{G}\mathbf{x} \preceq \mathbf{h}$$

$$\text{and} \quad \mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\preceq$ denotes "elementwise less than or equal to".

Your task is to formulate the SVM dual problems as a QP and solve it using `CVXOPT`, i.e. specify the matrices $\mathbf{P}, \mathbf{G}, \mathbf{A}$ and vectors $\mathbf{q}, \mathbf{h}, \mathbf{b}$.

```
In [3]: def solve_dual_svm(X, y):
            """Solve the dual formulation of the SVM problem.

            Parameters
```

```
    ----------
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).

    Returns
    -------
    alphas : array, shape [N]
        Solution of the dual problem.
    """
    # TODO
    # These variables have to be of type cvxopt.matrix
    P = matrix(np.dot(X, X.T) * y[None, :] * y[:, None])
    q = matrix(-np.ones(shape=(N, 1)))
    G = matrix(-np.eye(N))
    h = matrix(-np.zeros(shape=(N,1)))
    A = matrix(y[None, :])
    b = matrix(0.0)

    solvers.options['show_progress'] = False
    solution = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(solution['x'])
    return alphas
```

1.4 Task 2: Recovering the weights and the bias

```
In [4]: def compute_weights_and_bias(alpha, X, y):
            """Recover the weights w and the bias b using the dual solution alpha.

            Parameters
            ----------
            alpha : array, shape [N]
                Solution of the dual problem.
            X : array, shape [N, D]
                Input features.
            y : array, shape [N]
                Binary class labels (in {-1, 1} format).

            Returns
            -------
            w : array, shape [D]
                Weight vector.
            b : float
                Bias term.
            """
            w = np.dot(X.T, alpha * y[:, None])
            idx_sv = np.where(alpha > 1e-4)[0][0]
```

```
    b = y[idx_sv] - np.dot(X[idx_sv, :], w)
    return w, b
```

1.5 Visualize the result (nothing to do here)

```
In [5]: def plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b):
            """Plot the data as a scatter plot together with the separating hyperplane.

            Parameters
            ----------
            X : array, shape [N, D]
                Input features.
            y : array, shape [N]
                Binary class labels (in {-1, 1} format).
            alpha : array, shape [N]
                Solution of the dual problem.
            w : array, shape [D]
                Weight vector.
            b : float
                Bias term.
            """
            plt.figure(figsize=[10, 8])
            # Plot the hyperplane
            slope = -w[0] / w[1]
            intercept = -b / w[1]
            x = np.linspace(X[:, 0].min(), X[:, 0].max())
            plt.plot(x, x * slope + intercept, 'k-', label='decision boundary')
            # Plot all the datapoints
            plt.scatter(X[:, 0], X[:, 1], c=y)
            # Mark the support vectors
            support_vecs = (alpha > 1e-4).reshape(-1)
            plt.scatter(X[support_vecs, 0], X[support_vecs, 1], c=y[support_vecs], s=250, marke
            plt.xlabel('$x_1$')
            plt.ylabel('$x_2$')
            plt.legend(loc='upper left')
```

The reference solution is

```
w = array([[-0.69192638],
           [-1.00973312]])
```

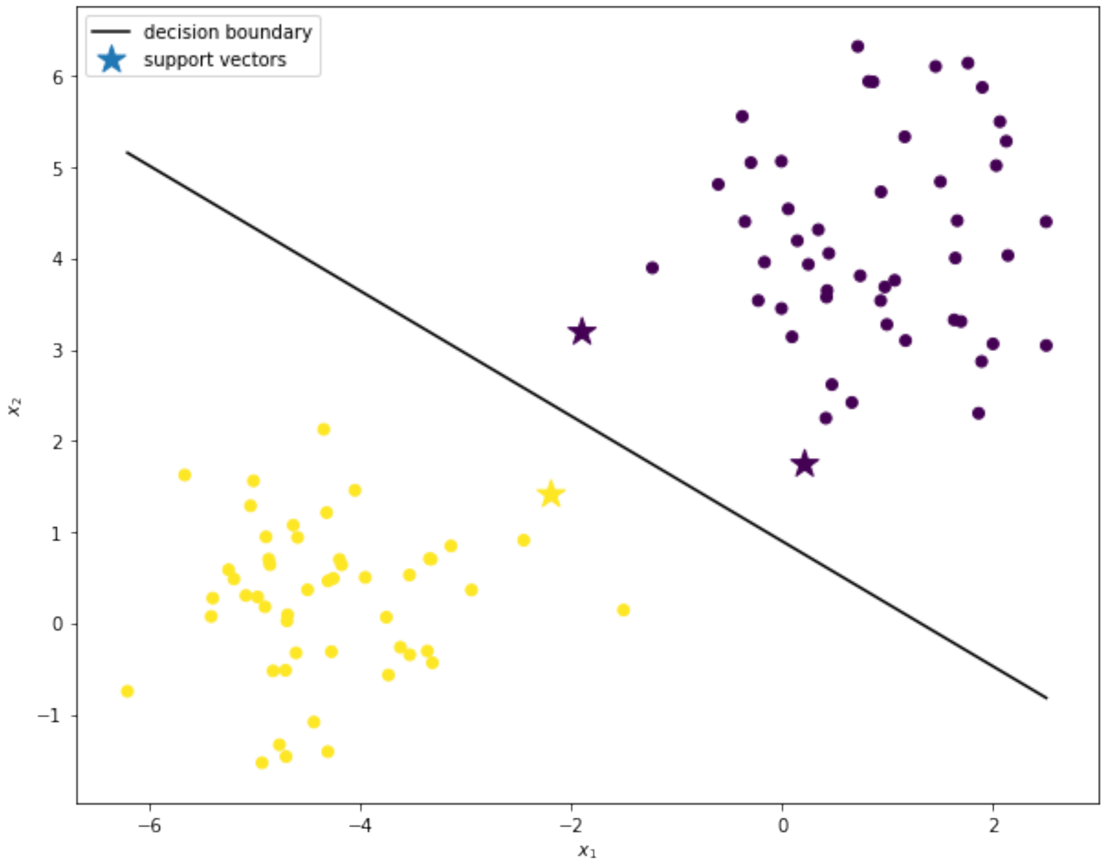
```
b = 0.907667782
```

Indices of the support vectors are

```
[38, 47, 92]
```

```
In [9]: alpha = solve_dual_svm(X, y)
        w, b = compute_weights_and_bias(alpha, X, y)
```

```python
plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b)
plt.show()
print('weights w = \n {} \n bias b = \n {}'.format(w, b))
print('Indices of the support vectors are {}'.format(np.where(alpha > 1e-4)[0]))
```




```
weights w =
 [[-0.69192638]
 [-1.00973312]]
 bias b =
 [ 0.90766782]
Indices of the support vectors are [38 47 92]
```